CPPC: Correctable Parity Protected Cache

Mehrtash Manoochehri, Murali Annavaram and Michel Dubois
Ming Hsieh Department of Electrical Engineering
University of Southern California, Los Angeles
{mmanooch, annavara}@usc.edu, dubois@paris.usc.edu

ABSTRACT
Due to shrinking feature sizes processors are becoming more vulnerable to soft errors. Write-back caches are particularly vulnerable since they hold dirty data that do not exist in other memory levels. While conventional error correcting codes can protect write-back caches, it has been shown that they are expensive in terms of area and power. This paper proposes a new reliable write-back cache called Correctable Parity Protected Cache (CPPC) which adds error correction capability to a parity-protected cache. For this purpose, CPPC augments a write-back parity-protected cache with two registers: the first register stores the XOR of all data written to the cache and the second register stores the XOR of all dirty data that are removed from the cache. CPPC relies on parity to detect a fault and then on the two XOR registers to correct faults. By a novel combination of byte shifting and parity interleaving CPPC corrects both single and spatial multi-bit faults to provide a high degree of reliability. We compare CPPC with one-dimensional parity, SECDED (Single Error Correction Double Error Detection) and two-dimensional parity-protected caches. Our simulation results show that CPPC provides a high level of reliability while its overheads are less than the overheads of SECDED and two-dimensional parity.

Categories and Subject Descriptors
B.3.2 [**MEMORY STRUCTURES**]: Design Styles—*Cache memories*

General Terms
Reliability

Keywords
Reliability, Cache, Parity

1. INTRODUCTION
Soft errors are a growing concern for microprocessor reliability. Since about 60% of the on-chip area is occupied by caches in today's microprocessors, caches have a considerable impact on microprocessor reliability. Most caches today are write-back caches. They contain dirty data which have no back-up copies in other memory levels, and therefore are highly vulnerable to soft errors. To protect against soft errors in write-back caches, two protection codes are common:

ISCA'11, June 4–8, 2011, San Jose, California, USA.

1- Single Error Correction Double Error Detection (SECDED): These codes are common in commercial processors [1, 11, 14, 18]. L2 and L3 caches are often protected by SECDED codes but SECDED codes are less attractive for L1 caches because they take a long time to decode [3] and may add some extra cycles to the Load latency in high-speed microprocessors. Furthermore, the area overhead of SECDED codes is high as it takes 8 bits to protect a 64-bit word, a 12.5% area overhead.

2- Parity: Because of the high overheads of SECDED codes, some commercial microprocessors protect L1 caches with parity bits at different granularities such as block [17], word [8] or byte [6]. Although parity bits are very effective in L1 write-through caches because they detect faults recoverable from the L2 cache, they do not provide any correction capability for the dirty data in L1 write-back caches. In some processors such as [8] in which the L1 write-back cache is protected by one parity bit per word, an exception is taken whenever a fault is detected in a dirty block and program execution is halted. Hence, even a single-bit error in a write-back parity-protected cache may cause the processor to fail.

A reliable and low-overhead scheme to correct errors in write-back caches is sorely needed in today's processors. This paper addresses this problem and introduces a new reliable write-back cache called Correctable Parity Protected Cache (CPPC) which augments a parity-protected cache with two registers to provide error correction capability. In order to reduce error correction overheads, CPPC increases the granularity of error correction (the number of bits which are protected together by an error correction code) from words or blocks to larger granularities up to the entire cache. To the best of our knowledge, this is the first paper to introduce this idea. By increasing the granularity of error correction, CPPC has much lower overheads than previously proposed schemes to correct both single-bit and spatial multi-bit faults.

CPPC has the following desirable features:

1. **Enlarging the protection domain efficiently**. Existing error correction codes are mostly attached to blocks or words (protection domains). Since error correction incurs both area and power penalties, it is best if the error correction domain can be enlarged to reduce the penalties. Furthermore, since the SEU (Single Event Upset) rate is very small, the impact on reliability of enlarging the protection domain is minuscule. Unfortunately, current methods do not enlarge the protection domain efficiently. For example, if an entire block is protected by ECC (Error Correction Code), a read-modify-write operation must be performed for every partial write to any block in the cache, with significant energy and performance impacts. CPPC enables efficient cache protection at any cache granularity, such as multiple blocks or even the entire cache, thereby saving area and other resources.

2. **Decoupling error detection and error correction**. Error detection can be done cheaply while correction is expensive. Hence, decoupling detection and correction is beneficial if errors are rare events, which is the case with soft errors. In CPPC, a lightweight detection code is checked during Loads and the correction procedure is invoked only when a rare error event is detected. Several prior studies also emphasized the importance of decoupling correction and detection [12, 19, 23].

3. **Decoupling the protection of clean and dirty data**. A large amount of cache data is clean and does not need error correction capability because the correct data exist in the lower level of the memory hierarchy. The efficiency of error correction can be improved by protecting dirty data only, as is done in CPPC. In [15] the authors exploit this decoupling idea as well. They protect dirty blocks by ECC, but, when a dirty block becomes clean, some ECC bits are gated off and ECC is converted to parity. In [13] the authors propose to decouple the protection of clean and dirty data in L2 and L3 caches by protecting dirty lines with a SECDED code and clean lines with parity bits. In this scheme, protection codes are saved in a separate array and the number of dirty lines is controlled by early write-back operations.

4. **Protecting against spatial multi-bit errors**. The number of spatial multi-bit errors (spatial MBEs) increases as the feature size decreases [16], because a single particle strike can flip more bits when the area occupied by each bit is less. Thus, a reliable cache protection scheme must recover from spatial MBEs. Physical bit interleaving, in which bits of different words are stored in adjacent cells, is a common technique to detect and correct spatial MBEs since it converts a spatial multi-bit error into several single-bit errors in different words. Bit interleaving tolerates spatial multi-bit faults at the expense of increased energy consumption, as was shown in [12]. CPPC can correct spatial multi-bit errors, without the high power cost associated with bit interleaving.

The basic CPPC detects errors with parity bits and corrects errors in clean data by re-fetching the data from the next level of the memory hierarchy. To provide fault-correction capability for dirty blocks, a classical cache design is augmented with two special-purpose registers. The first register keeps the XOR of all data written to the cache while the second register keeps the XOR of all dirty data removed from the cache either when replacing the dirty data or when overwriting the dirty data. When a fault is detected in a dirty block, CPPC recovers from the fault by XORing the two registers which is equivalent to computing the XOR of all dirty words currently in the cache, and then XORing this result with all current dirty words in the cache except for the faulty data. The net result of this operation is the correct value of the data detected as faulty.

We further enhance the basic CPPC to recover from spatial MBEs with a novel combination of byte shifting and parity interleaving and with negligible additional overhead over the basic CPPC.

A CPPC can be a cache at any level of the cache hierarchy. In this paper we focus on L1 and L2 caches. We evaluate both L1 and L2 CPPCs with the help of SimpleScalar [4] and CACTI [5]. An L1 CPPC is highly reliable, at the cost of a small amount of area and energy overheads. We expect CPPCs to be even more efficient as lower-level caches. In particular, we show that the energy consumed by an L2 CPPC is only 7% higher than the same cache with parity protection, which is very small, considering that a CPPC provides error correction capability for dirty data. We expect the energy overhead of an L3 CPPC to be even less.

We also compare CPPCs to caches protected by SECDED combined with physical bit interleaving and by two-dimensional parity. Our simulation results show that the dynamic energy consumption of an L2 CPPC is lower by 46% and 49% as compared to L2 caches protected by SECDED and two-dimensional parity, respectively.

The rest of the paper is organized as follows. Section 2 covers the related work. Section 3 describes the basic CPPC architecture. In Section 4, the basic CPPC is extended to cover spatial multi-bit faults. The impacts of CPPCs' mechanisms on various metrics are discussed and evaluated in Sections 5 and 6. Finally, Section 7 provides our conclusions and proposes future work.

2. RELATED WORK

Protecting caches with two-dimensional parity was proposed in [12]. In this approach the horizontal parity (along a row) detects errors and the vertical parity (along a column) corrects them. In this scheme, spatial multi-bit errors are corrected by interleaving data array rows among vertical parity rows, so that adjacent data array rows are protected by different vertical parity rows. Since the vertical parity of the cache changes on every Store, the new vertical parity must be computed by reading the old data from the cache and XORing it with the old vertical parity. This operation is called "read-before-write" and is done on every Store. Two-dimensional parity (or row-diagonal parity) is also used in RAID 6 (Redundant Array of Inexpensive Disks). However, unlike in RAIDs, the two-dimensional parity in caches requires that the read-before-write operation is executed for every cache miss in addition to all Stores. Furthermore, in the case of a miss, an entire cache line must be read. The energy cost of these read-before-write operations is therefore high.

To mitigate the area overhead of error protection codes in a last level cache, the codes could be saved in main memory, as was proposed in [23]. This scheme facilitates the implementation of very strong protection codes at the cost of more memory accesses. In ICR (In Cache Replication) [24] cache lines that have not been accessed for a long time are allocated to replicas of dirty blocks. ICR essentially trades off reduced effective cache size for better reliability. Thus the miss rate of the cache may be higher or, alternatively, dirty blocks may be left unprotected. In [25] a variant of ICR in which a small cache keeps a redundant copy of dirty blocks is proposed, but this scheme is not area-efficient for large caches. Several papers [2, 15] advocate early write-backs in order to increase the reliability of write-back caches. These schemes reduce the total number of dirty blocks and therefore the vulnerability of L1 caches to soft errors. Their energy consumption is high, especially when data locality is low and the number of write-backs is large. In [20] a small cache for saving check bits or replicas is proposed. In this scheme, a large amount of the dirty data remains unprotected if data locality is low.

3. BASIC CPPC

In this section, we first describe the architecture of the basic CPPC. As mentioned earlier, a CPPC can be a cache at any level of the cache hierarchy. However, to keep the description simple, we first describe the necessary hardware modifications to an L1 cache and the operation of a basic L1 CPPC. Then, later, in Section 3.5, we describe the modifications necessary for an L2 CPPC.

Figure 1 shows the components of a basic L1 CPPC. An L1 CPPC keeps one dirty bit per word in the cache tag array. A basic L1 CPPC also has at least one parity bit per word. In addition, an L1

CPPC has two registers, R1 and R2, each the size of a word. Register R1 keeps the XOR of all data words stored into the L1 cache and register R2 keeps the XOR of all dirty data words removed from the cache. Thus, R1 is updated on every Store to the cache. Dirty data are removed from the cache in two cases: 1) a Store is executed to a dirty word in the cache or 2) a dirty cache block is written back to the next level. In both cases the dirty data words that are overwritten or written-back are XORed into R2. At any time, the XOR of R1 and R2 is therefore equal to the XOR of all dirty words remaining in the cache.

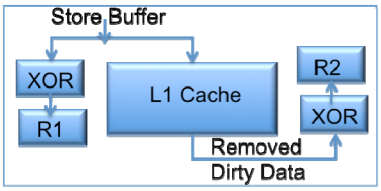


Figure 1. CPPC Structure

3.1 Normal L1 CPPC operation

During normal CPPC operation (no fault is detected) every Load checks the parity bit(s) associated with the word. On every Store, the new data is XORed into R1 in parallel with writing into the cache. When a Store updates an already dirty word, it is necessary to first read the old word before storing the new word. To implement this read-before-write, the cache operation on a Store is slightly different from a regular cache. On every Store, the per-word dirty bit is first checked in conjunction with the tag comparison. If the dirty bit is set the old data is read and XORed with R2. If the dirty bit is not set, the Store proceeds normally. Note that even in a regular cache every Store must first check the tags to confirm a hit before writing data. On a byte Store, the new byte is XORed with the corresponding byte of R1 based on the address bits and the old byte is XORed with the corresponding byte of R2, if the word in the cache is dirty. Figure 2 shows the write operation in a basic CPPC.

A few microarchitectural modifications needed in a CPPC are worth mentioning. The read of old data and write of new data can be done in parallel if the cache has separate read and write ports, which is widespread in modern processors. Thus some Stores in a CPPC contend for the read port. Since Stores access the read port *only* when the cached data is already dirty, this contention does not always happen. These unpredictable conflicts with Loads may be problematic for L1 caches in modern out-of-order processors, in which a Load and its dependent instructions are speculatively scheduled by assuming a fixed L1 cache hit latency. Due to port contention the Load latency may be variable, causing costly replays of the Load and its dependent chain of instructions. To solve this problem, a Store in an L1 CPPC must steal cycles from the read port. Cycle-stealing can be effectively done by coordinating the Store buffer with the Load/Store scheduler which keeps track of reservations of the cache read port and can inform the Store buffer when it is safe to access the read port to avoid potential contention with incoming Loads. Such a design eliminates much of the variability in Load hit latencies due to CPPC.

Finally, on each cache write-back, all dirty words of the evicted block must be XORed into R2. Since write-back caches typically process write-backs through a victim buffer, this operation can be done "in the background", off the critical path, and without any significant performance impact.

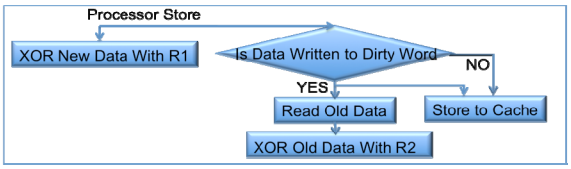
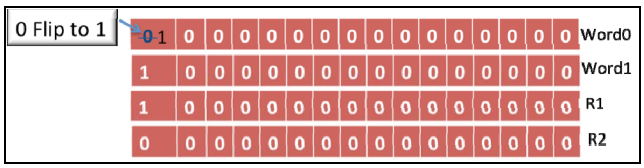


Figure 2. Write Operation in CPPC

3.2 Recovery

The normal operation of a basic L1 CPPC does not add significant overhead to the operation of a regular cache, but when an error is detected, a complex recovery algorithm is triggered.

When a fault is detected in a clean word, it is converted to a miss and the cache block is re-fetched from the next level of the memory hierarchy. To recover from a fault in a dirty word, the recovery algorithm first XORs R1 and R2 together resulting in the XOR of all dirty words of the cache. The XOR of R1 and R2 is then XORed with all dirty words currently in the cache *except* the faulty word. The resultant word is the correct value of the faulty word. Note that the recovery operation requires accessing *every* cache block to find the dirty words. This operation is time-consuming and complex but extremely rare. Recovery could be done in hardware by a separate micro-engine associated with the cache controller, but a better implementation of the basic CPPC recovery algorithm is probably to trigger an exception in the processor and execute a software handler as was proposed in [7]. Whatever the mechanism implementing the recovery, the effects of recovery on performance and energy consumption can be ignored, because it is an extremely rare event.



Bit0_Word0=XOR(bit0_word1, bit0_R1, bit0_R2)=XOR(1, 1, 0)= 0

Figure 3. Error Recovery in the Basic CPPC

3.3 Illustrative example

In this section, we illustrate the basic CPPC with an example. Figure 3 shows a cache with two words labeled Word0 and Word1, and registers R1 and R2. For simplicity, assume that each word is 16 bits wide and the contents of the cache and registers are initially zero. The processor first stores 0x0000 into Word0. This value is XORed into R1. The processor then stores 0x8000 into Word1. This value is XORed into R1 and the contents of R1 is 0x8000. Throughout this process no data has been replaced and hence the value of R2 is unchanged. Now consider when a particle strike flips the Most Significant Bit (MSB) of Word0 from 0 to 1. The parity checker detects the fault on a Load of Word0 at which time fault recovery is activated. The recovery algorithm XORs R1 and R2 and then XORs the result with every word in the cache except for the faulty word. In this example, R1, R2 and Word1 are XORed. The result is the correct value of Word0.

3.4 Enhancing fault tolerance

The basic CPPC corrects at least one fault in dirty words and an odd number of temporal faults in clean words. For example, with one parity bit per word and two 64-bit registers to protect dirty data, a basic L1 CPPC corrects all odd numbers of faults in a dirty word provided there are no faults in other dirty words. Thus, error correction is added to error detection with a very small area

overhead. The error correction capability of a CPPC can be scaled up in two different ways.

First, we can increase the number of parity bits per word. For instance, with eight parity bits per word (i.e., one parity bit per byte), the correction capability is enhanced eight times. In this situation, the cache corrects an odd number of faults provided they occur in different bytes of different words. For example, three faults in byte 1 of a dirty word and five faults in byte 2 of another dirty word can be corrected. Hence, the granularity of correction goes from all words to all bytes.

Second, the XOR of dirty words can be maintained in smaller granularities than the entire cache. For example, instead of having only two registers R1 and R2, we can have four registers. R1, R2, R3 and R4. R1 and R2 maintain the XOR of dirty data for one half of the cache, and R3 and R4 maintain the XOR of dirty data for the other half of the cache. With this improvement, the protected domain is cut in half and the correction capability is improved accordingly. Hence, the correction capability can be scaled up to achieve a desired level of reliability simply by adding more register pairs.

3.5 CPPC as an L2 cache

The basic CPPC described above in the context of an L1 cache can be easily implemented in any level of the cache hierarchy. If an L1 cache is a CPPC, then the size of both R1 and R2 is a word since the processor updates L1 at the granularity of words. If an L2 cache is a CPPC, then R1 and R2 must keep track of dirty data at the granularity written from L1 to L2. R1 and R2 must have the size of an L1 cache block. An L2 CPPC must also maintain one dirty bit per unit of L1 cache block size.

3.6 Correcting spatial MBEs with interleaved parity in CPPC

With the soon-expected predominance of spatial MBEs, it is desirable to upgrade the basic CPPC with interleaved parity in order to detect and correct a subset of spatial MBEs. Since regular parity only detects an odd number of errors, they are unable to detect an even number of errors such as spatial two-bit errors where a single strike flips two adjacent bits. The basic CPPC described above can be upgraded by using interleaved parity bits to detect spatial multi-bit errors. Interleaved parities are the XOR of non-adjacent bits in a protection domain. For example, the 8-way interleaved parity bits in a word are computed by the XORs of bits whose distance is eight (Parity[i] = XOR (data_bit [i], data_bit [i+8], …, data_bit [i+56]). With interleaved parity, every spatial MBE which flips 8 or fewer bits in a word can be detected. The basic CPPC using interleaved parity can correct some but not all detected spatial MBEs. For example, a detected horizontal multi-bit fault can be corrected in the basic CPPC with interleaved parity. A horizontal multi-bit fault occurs either in one word or across the boundary of two words. If only one word is faulty, the basic CPPC with interleaved parity can generate the corrected word after recovery without any change. If the fault occurs across the boundary of two words, it is also correctable because *different* parts of the two words are affected by the fault. For instance, with 8-way interleaved parity bits in a 64-bit word, if a 7-bit horizontal spatial fault occurs across the boundary of two words such as for bits 62-63 of the left-side word and bits 0-4 of the right-side word, parity bits P6-P7 of the left-side word and parity bits P0-P4 of the right-side word detect the fault and since *different* bytes of *different* words are faulty this fault involving 7 bits can be recovered from.

Unfortunately the basic CPPC with interleaved parity cannot correct vertical multi-bit errors. Motivated by the fact that spatial MBEs will become prevalent in the near future, we now introduce enhancements to the basic CPPC with interleaved parity which can detect and correct spatial MBEs, both horizontal and vertical.

4. SPATIAL MULTI-BIT FAULT TOLERANT CPPC

In this section, we propose two enhancements to the basic CPPC in order to provide spatial MBE correction capability in both dimensions. The two enhancements we propose are 1) byte-shifting or 2) adding register pairs. At first we expound on the byte-shifting idea, and then, in Section 4.11, we explain how complex byte-shifting operations can be eliminated by adding register pairs at the cost of more hardware.

Again, to simplify the presentation, we assume in the following that the write granularity is a 64-bit word (as in an L1 CPPC) to show how the technique works. We also make the assumption that a spatial MBE can *only* occur in an 8-by-8 bit square. The same designs with obvious modifications can be deployed in general to correct spatial MBEs occurring in an N-by-N bit square.

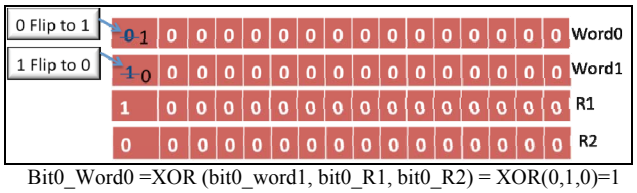
4.1 Vertical spatial multi-bit errors

Although the basic CPPC with interleaved parity can correct horizontal spatial multi-bit errors, it cannot correct vertical multi-bit errors. For example, a two-bit vertical fault which flips the first bits (bit 0) of two vertically adjacent dirty words cannot be corrected in the basic CPPC with interleaved parity because bit 0 of R1 is produced by the XOR of these two faulty bits. Thus, we cannot recover the original values of these two faulty bits using R1 and R2. However, if we were to XOR vertically adjacent bits into *different* bits of R1 and R2 instead of the *same* bits, we could potentially recover from vertical multi-bit errors. This is the major intuition leading to our byte-shifting technique. In order to XOR vertically adjacent bits into different bits of R1 and R2, we propose to rotate data *before* XORing them into R1 and R2 using different amounts of rotation for any two adjacent rows. Note that the data is rotated just before the XOR is performed into R1 and R2 but the data stored in the cache is NOT rotated.

4.2 Illustrative example

Figure 4 illustrates why the basic CPPC with interleaved parity fails to correct vertical spatial MBEs. This example starts with the same assumptions and initial conditions as the example of Figure 3. Consider the case that a single particle strike flips two bits that are vertically adjacent after the processor has executed two Stores. In this example, the particle strike flips the MSB of each of the two words, Word0 and Word1. When the processor reads Word0 the basic CPPC fault recovery mechanism XORs Word1 with R1 and R2. However, since Word1's MSB is also faulty the XOR operation produces a 1, instead of the correct value, 0. Hence, the equation for error correction shown in the figure fails and the basic CPPC cannot correct the vertical 2-bit fault.

To solve this problem, in Figure 5, we rotate Word1 by one byte to the left before XORing it with R1. Note that the data stored in the cache is NOT rotated. Therefore, bit j of R1 is derived from the XOR of bit j of Word0 and bit (j+8) mod 16 of Word1. With this approach a vertical fault in bit 0 of the two adjacent words can be corrected. The fault recovery equations are modified as shown in the figure to account for the byte rotation. As the equations of Figure 5 show faulty bits can be corrected using byte shifting.



Bit0_Word0 =XOR (bit0_word1, bit0_R1, bit0_R2) = XOR(0,1,0)=1

Figure 4. Basic L1 CPPC Fails to Correct Vertical MBEs

Bit0_Word0 = XOR (bit8_word1, bit0_R1, bit0_R2) = XOR(0,0,0) = 0 (ERROR CORRECTED!)

Bit0_Word1 = XOR (bit8_word0, bit8_R1, bit8_R2) = XOR(0,1,0) = 1 (ERROR CORRECTED!)

Figure 5. Byte Shifting to Correct Vertical MBEs

4.3 Byte-shifting operation

The byte-shifting technique illustrated above can easily be added to the basic CPPC with interleaved parity. The basic CPPC with interleaved parity can correct horizontal multi-bit faults. With byte-shifting, vertical multi-bit faults can now be corrected as well so that spatial multi-bit faults are correctable.

Figure 6 shows a data array in which every 64-bit word is augmented with 8-way interleaved parity bits that can detect up to an 8-bit error in each word. To rotate data, a barrel shifter is added to the basic CPPC to XOR new data into R1. The same structure exists for R2 and is not shown in the figure. Some bits of the word address are connected to the control input of the barrel shifter. For instance, in Figure 6 three bits of the Store address specify eight separate amounts of rotation for eight different data array rows. All data array rows that are shifted by the same amount are lumped into a *rotation class*. All rows that belong to the same rotation class have their bytes aligned. With 8-way interleaved parity, we create eight rotation classes to correct all spatial multi-bit faults contained in an 8*8 square.

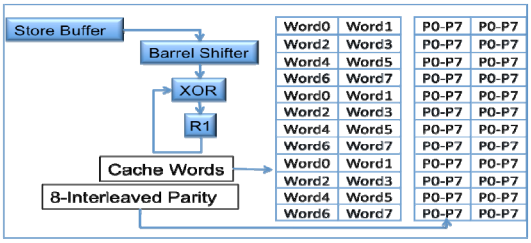


Figure 6. CPPC With Barrel Shifter

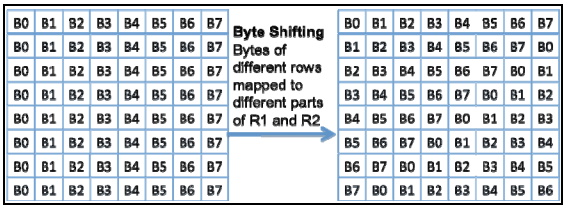


Figure 7. Byte Shifting in an 8*8 Array

Figure 7 shows two data arrays. The array on the left side shows the arrangement of bytes in the cache and the array on the right side shows how different bytes of different rows are XORed to R1 and R2 after byte-shifting. As Figure 7 shows every multi-bit fault contained within an 8*8 square in the left-side array affects different bits of R1 and R2 after rotation as illustrated in the right-side array. For example, if a 3-bit vertical fault occurs in the first bit of byte 0 of the first three rows (left array), the fault affects bit 0 of byte 0, bit 0 of byte 6 and bit 0 of byte 7 of R1 and R2 (right array). Therefore, by adding two registers and two barrel shifters to a parity-protected cache with 8 interleaved parity bits, CPPC has the potential to correct all spatial MBEs contained in an 8*8 square.

4.4 Dirty data error recovery

The correction of a spatial multi-bit error in a CPPC is complex. However, error recovery is invoked extremely rarely and so its complexity is not an issue. To recover from an error after its detection when reading a word, a CPPC activates a step-by-step procedure as follows.

Step 1. XOR all dirty words in the cache after rotation except for the faulty word. Then, XOR that result with both R1 and R2. If a fault is detected in any other dirty words during this step then go to step 3.

Step 2. Rotate the result of step 1 in reverse, right instead of left, then copy it into the faulty word to recover from the error. Go to step 7.

Step 3. Identify the parity bits which detected the faults in different words. If some faulty parity bits are common among faulty words, go to step 5.

Step 4. In this step, faulty parity bits are different in different dirty words. For every faulty parity bit, repeat the following operation:

XOR the data protected by the faulty parity bit with all other dirty words in the cache except for the one which is being corrected. Then XOR this result with R1 and R2 and copy this result into the faulty word.

All errors are recovered from. Go to step 7.

Step 5. Check the distance between the rows of faulty words. If the distance between any such pairs of rows is more than 8 the error is not correctable because it affects an area larger than the correction capability of the cache (8x8). Go to step 7.

Step 6. The error may be correctable. The CPPC recovery procedure assumes that this is a spatial multi-bit fault and invokes the fault locator (Section 4.5) to find the location of faults. The fault locator finds which bit(s) of data protected by a parity bit is flipped. This function is needed because parity bits only detect errors but do not point to the exact location of faulty bits.

If the fault locator can find the location of faulty bits, correct them. Otherwise, the errors cannot be corrected.

Step 7. If errors were corrected, continue the normal execution of the program. Otherwise, halt the program and raise a machine-check (fatal) exception (DUE).

4.5 Fault locator

When a multi-bit fault occurs, the CPPC recovery procedure must find the position of the flipped bit(s). Parity bits just indicate whether one or more bits are faulty out of several bits but do not point to the position of the faulty bits. Finding the location of faulty bits is the function of the CPPC's fault locator.

Since every bit of all words is XORed into a bit of R1 and R2, the fault locator can use these two registers to find the location of faulty bits. The fault locator first XORs all dirty words in the cache including all faulty words after proper rotation and then it XORs this result with R1 and R2. This 64-bit result is called R3 in

this section, although R3 is not necessarily a register. In effect, R3 is the XOR of a value with itself. In the absence of faults, all bits of R3 would be zero. However, in the presence of faults, some bits of R3 are set in the locations where faulty bits are XORed. For example, if there is a two-bit vertical fault in bit 0 of the first two words of Figure 7, R3 has two 1s in bit 0 and bit 56 positions and all other bits are 0.

When the CPPC fault locator is invoked, it knows the parity bits which detected errors, the rotation classes of the faulty words and the locations of bits in R3 whose values are 1. These three critical pieces of information are then exploited to find the location of faulty bits. For example, in the two-bit fault example of the above paragraph, the only bits of R3 to have two 1s are bit 0 and bit 56. At the same time, the parity bits P0 of the rotation class 0 word and P0 of the rotation class 1 word have detected the two faults. Therefore, the fault locator uses these three pieces of information and concludes that this is a 2-bit fault in bit 0 of those two words.

An error contained in an 8*8 square occurs either in the *same* byte of different rows or across the boundary of the *same* two adjacent bytes of different rows. If the faulty byte or the faulty adjacent two bytes are located, the error is correctable because the faulty parity bits and the bits set in R3 point to the bits of every byte that have been flipped. For example, if the fault locator finds that byte 0 of some words contains faults, the error is located because parities show the exact location of the faults. In general, when a correctable fault has been detected, the following step-by-step procedure is executed to locate faulty bits.

Step 1. Identify the bytes of R3 which are non-zero. We call them *R3 faulty bytes*.

Step 2. For every R3 faulty byte, identify the bytes of all faulty words which have been XORed into that R3 faulty byte. These sets of bytes are called *faulty sets*.

Step 3. Inspect all faulty sets. If there is only one common member in all faulty sets, all faults must have occurred in that common byte of all faulty words. The fault is located and can be corrected, thus go to step 5. If there is no common member in the faulty sets, check whether all faulty sets include at least one of two adjacent bytes. If such a pair of adjacent bytes cannot be found, the error is not correctable, go to step 5.

Step 4. Categorize R3 faulty bytes based on the two known bytes and remove all other bytes from R3 faulty sets. At this time, the faulty set associated with one R3 faulty byte contains at most two known located bytes. Find an R3 faulty byte whose faulty set includes only one of the two bytes. For this R3 faulty byte, bits set to 1 in R3 show the exact location of errors in the byte which is the only member of that faulty set. Thus, the location of errors in one byte of one of the faulty words is found. In this situation, other errors in that faulty word occurred in the other byte. Thus, errors in one word are completely located. Then remove the bytes of that corrected word from the faulty sets, reset the bits of R3 which are 1 due to this faulty word. Repeat this step until all faulty bits are located in faulty words.

Step 5. If faults were located, send the located words to the cache in order to correct bits. Otherwise, halt the program and raise a machine-check exception (DUE).

To illustrate the procedure, assume that parity bits P0-P7 of the first four rows belonging to the first four rotation classes (Figure 7) have detected errors and bits 0-12 and 45-63 of R3 are 1 and all other bits of R3 are 0. In this case, the steps taken in the above algorithm are as follows.

Step 1. Byte 0, Byte 1, Byte 5, Byte 6 and Byte 7 of R3 are faulty.

Step 2.

The faulty set of byte 0 of R3 is {Byte 0, Byte 1, Byte 2, Byte 3}

The faulty set of byte 1 of R3 is {Byte 1, Byte 2, Byte 3, Byte 4}

The faulty set of byte 5 of R3 is {Byte 5, Byte 6, Byte 7, Byte 0}

The faulty set of byte 6 of R3 is {Byte 6, Byte 7, Byte 0, Byte 1}

The faulty set of byte 7 of R3 is {Byte 7, Byte 0, Byte 1, Byte 2}

Figure 8 illustrates the faulty sets identified in step 2.

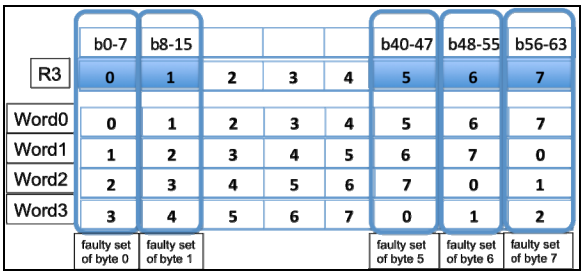


Figure 8. Faulty Sets Obtained in Step 2

Step 3. There is no common byte in all faulty sets but all faulty sets contain either byte 0 or byte 1. Thus, faults are across the boundary of bytes 0 and 1.

Step 4. After removing all bytes besides bytes 0 and 1 from all faulty sets, the faulty sets corresponding to each R3 faulty byte are reduced to:

Byte 0 of R3: {Byte 0 of class 0, Byte 1 of class 1}

Byte 1 of R3: {Byte 1 of class 0}

Byte 5 of R3: {Byte 0 of class 3}

Byte 6 of R3: {Byte 0 of class 2, Byte 1 of class3}

Byte 7 of R3: {Byte 0 of class 1, Byte 1 of class2}

Figure 9 illustrates the reduced faulty sets at the start of step 4.

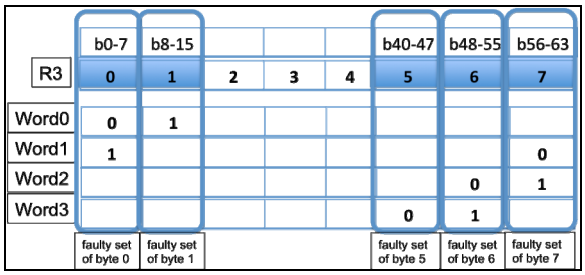


Figure 9. Reduced Faulty Sets at the Start of Step 4

The procedure starts with byte 1 of R3, whose faulty set only has one member. Since byte 1 of class 0 word is aligned with byte 1 of R3, the bits set in this R3 byte point to the exact location of the faulty bits in that byte. As the first 5 bits of byte 1 of R3 are 1, the first 5 bits of byte 1 of the class 0 word are faulty. Because these 5 faults bits are detected by parity bits P0-P4 of the class 0 word, and parity bits P5-P7 also detect the presence of faults, the faults detected by P5-P7 must be due to bit flips in bits 5-7 of byte 0 of the class 0 word. Thus, faults in the class 0 word are located.

Since faults in the class 0 word are located, the procedure removes the bytes of the class 0 word from all the faulty sets. Then the bits

of R3 which are 1 due to the faults in class 0 word are reset i.e. bits 5-12 of R3 are reset. Thus, at this time bits 0-4 and 45-63 of R3 are 1 and all other bits are 0.

Now, we have the following (further) reduced faulty sets.

Byte 0 of R3: {Byte 1 of class 1}

Byte 5 of R3: {Byte 0 of class 3}

Byte 6 of R3: {Byte 0 of class 2, Byte 1 of class 3}

Byte 7 of R3: {Byte 0 of class 1, Byte 1 of class 2}

The fault locator repeats the procedure which was used to locate faults in the class 0 word and finds the location of faulty bits in words from classes 1, 3 and 2, iteratively.

Step 5. The locator concludes that the spatial multi-bit fault occurred in bits 5-12 of 4 words from classes 0-3. Thus, the spatial multi-bit fault is located and is sent to the cache for correction.

An important issue here is whether the fault locator can find all errors in the correctable range. Most specifically, can several separate multi-bit faults have the same faulty parities, the same classes of faulty words and the same R3 pattern? This question is answered in Section 4.6.

4.6 Spatial multi-bit error coverage

With registers R1 and R2 in the CPPC of Figure 6, all spatial multi-bit faults are locatable by the fault locator, except for some special cases. For instance, in the case of an 8*8 fault, all parity bits of all 8 words detect errors and all bits of R3 are set. Hence, there is no way to figure out which 8 bits of different words are faulty. Another situation is when a spatial multi-bit fault does not affect adjacent rows and only affects rows distant by 4 rows from each other. For example, a fault which flips bits from byte 0 of a class 0 word and from byte 0 of a class 4 word cannot be located. This is because the content of R3 is identical in both cases and the fault locator cannot figure out whether the fault has occurred in byte 0 or in byte 4 of the two faulty words. These errors remain DUEs and the probability of such events is very small.

Nevertheless, to solve these two problems, we can add another pair of registers so that the first four rotation classes (classes 0-3) are protected by one pair and the other four rotation classes (classes 4-7) are protected by the other pair. In this way, all multi-bit faults are locatable. By using two register pairs, an 8*8 error is converted to two separate 4*8 errors in different pairs and they are correctable. Hence, the CPPC of Figure 6 needs to be augmented with two pairs of registers to correct the two fault patterns above. This is trade-off between area and spatial multi-bit error correction capability in CPPC designs.

4.7 Aliasing with temporal multi-bit errors

The byte-shifting scheme of CPPC supposes that if there are several faults in adjacent rows within the expected correctable range, it is a spatial multi-bit fault. However, it is possible — although extremely unlikely-- to have several temporal single-bit faults in adjacent rows. If there are several temporal faults in adjacent rows, they might be incorrectly detected as a spatial multi-bit fault which causes the fault locator to produce an incorrect output. When the fault locator produces a wrong result, some correct bits are flipped incorrectly. For example, in Figure 7, if we have two temporal errors in bit 56 of the class 0 word and in bit 8 of the class 1 word, the fault locator decides incorrectly that bits 0 of both words are faulty. Instead of a two-bit error, we end up with a 4-bit error! Furthermore, to make matters worse, a 2-bit DUE is converted to a 4-bit SDC (Silent Data Corruption).

For this to happen, after the first fault occurs, the second fault must happen in one of seven bits, out of all the bits in the cache, in a very short period of time, i.e., before the correction of the first fault. For instance, a fault in bit 56 of a class 0 word must be followed by a second fault in bit 0 of class 1 word or bit 8 of class 2 word or bit 16 of class 3 word or bit 24 of class 4 word, … or bit 48 of class 7 word, in order for the locator to confuse the temporal MBE for a spatial MBE.

This problem can be mitigated by increasing the number of register pairs. If we have two pairs of registers in which each pair is responsible for four separate rows, the probability of correcting a temporal multi-bit fault as a spatial multi-bit fault decreases by half (if the first fault is in a class 0 word, the second must now be in classes 1-3). Therefore, after a single-bit fault, the second one must occur in one of 3 bits out of all the bits of the cache in a very short period of time. With four pairs of registers so that each pair protects two classes of words, this problem is mitigated further so that after the first fault, the second fault must occur in one specific bit. Finally if we use 8 pairs of registers, this problem is completely eliminated. In this case, byte-shifting is also eliminated, and all multi-bit faults can be corrected. This radical solution is explained further in Section 4.11.

Based on our evaluations, the mean time to have one such mistake in an L2 cache with one pair of registers with the specifications used in Section 6 is $4.19 * 10^{20}$ years, which is 5 orders of magnitudes larger than DUEs due to temporal 2-bit faults. Consequently, it is an extremely rare event and its effect on cache reliability is negligible.

4.8 Barrel shifter implementation

Our basic byte-shifting technique uses two barrel shifters to rotate the location of bytes before XORing into R1 and R2, in order to provide spatial multi-bit error correction capability. In general, a barrel shifter is made of $\log_2(n)$ levels (n is the number of input bits) of multiplexers to rotate data. Fortunately, the barrel shifters in a CPPC have a very simple structure and rotate left only and by multiples of bytes only. Therefore, the barrel shifters of a CPPC only need $n/8 * \log_2(n/8)$ multiplexers and $\log_2(n/8)$ stages, which is significantly less than $n * \log_2(n)$ and $\log_2(n)$ in regular barrel shifters.

The time and energy required to rotate a 32-bit word in a barrel shifter was computed in [9] for a 90nm technology. The delay and energy consumption of rotating 32 bits are reported to be less than 0.4ns and about 1.5 pj, respectively. We used CACTI 5.3 to estimate the access latency of an 8KB direct-mapped cache. CACTI's estimate for the cache access time is 0.78ns, which is much longer than the delay of the barrel shifter. Thus, the barrel shifter operation is not in the critical path.

The energy consumption of the barrel shifter is also negligible compared to the cache as it consumes at most 1.5 pj [9]. However, the energy consumption of a 32KB, 2-way set-associative cache in the same 90nm technology, is estimated by CACTI to be 240 Pj per access. Consequently, the barrel shifter of CPPC has no effect on performance and energy consumption of the cache.

4.9 Design issues for R1 and R2 registers

If a CPPC has one write port, there is at most one XOR operation with R1 at a time. If the latency to rotate and XOR a new word

into R1 is shorter than the latency of writing into the cache, the two operations are overlapped. R2 is also XORed with removed dirty data due to write-backs and Stores into dirty words. Because the access time of the cache is larger than the time needed to rotating and XORing with R2, there will be no performance overhead except for port contention.

New or removed dirty data are XORed with R1 and R2 bit by bit and the XOR operation can be implemented by one level of XOR gates whose delay is very small as compared to the latency of an access to the cache as shown in Section 4.8. Hence, overall, operations on R1 and R2 are not in the critical path. If a cache has more ports, more registers and barrel shifters can be added to avoid performance overhead.

Another issue that might be raised is the possibility of a fault in either R1 or R2. Several solutions are possible. One possibility is to protect the registers by a correction code like SECDED. Another possibility is to protect registers with parity bits and check parities before each XOR operations whenever R1 and R2 registers are read. If a fault is detected by a parity bit, it can be recovered by XORing all the dirty words of the cache provided there is no fault in the dirty words of the cache.

4.10 Byte-shifting advantages

The main advantage of the byte-shifting technique is that it is area efficient as spatial multi-bit error correction is provided by adding several barrel shifters which are made with simple multiplexers.

Like in Section 3.4, in which we had a trade-off between area and reliability against temporal single-bit errors, here we have a trade-off between area and reliability against spatial multi-bit errors. If we intend to save area as much as possible, we can use two registers and correct all multi-bit errors except for some special cases. To increase the reliability against both spatial and temporal errors, we can also add more register pairs.

4.11 Spatial multi-bit error correction with more pairs of registers instead of byte-shifting

Another method to correct spatial multi-bit errors in a CPPC is adding more pairs of registers such that adjacent vertical rows are interleaved among different pairs of registers. For example, to correct all 8*8 errors in the cache of Figure 6, we use 8 pairs of registers, such that rows within distance of 8 (8 classes) are protected by different register pairs. In this case there is no byte-shifting, no barrel shifters, and the dirty words in every class are protected by a register pair associated with the class. With this approach, the resulting CPPC has some similarities with two-dimensional parity caches in that both correct spatial multi-bit errors in similar ways. However, there are significant differences. A CPPC hashes all bits of vertical rows into two words in an L1 cache or two entries with L1 block size in an L2 cache. Moreover, CPPC decouples the protection of dirty data from clean data and the pairs of registers only protect dirty data. Furthermore, CPPC executes the read-before-write operation only for writing in dirty data but two-dimensional parity executes this operation for all Stores and all cache misses. Finally CPPC saves a significant amount of energy as compared to two-dimensional parity. This is shown in Section 6.2.

5. IMPACTS OF CPPC'S RELIABILITY MECHANISMS ON VARIOUS METRICS

In CPPCs the recovery and locator algorithms are invoked extremely rarely and, because of this, its complexity and performance/energy overheads are not important, provided the algorithms can be implemented at low cost. The complex recovery/locator algorithm could be executed for example by a simple finite-state machine dedicated to memory reliability or to system reliability in general. Or, even more cheaply, it could also be implemented by special software exceptions. Reliability-Aware Exceptions (RAEs) have been proposed [7] as a simple and economical means to improve the reliability of processors. In any case, we do not discuss the implementation of the recovery/locator algorithm in the rest of this paper, but, rather, we focus on the overheads caused by CPPC's reliability mechanisms in the common case, i.e., in the absence of faults.

The correction capability of CPPC against temporal multi-bit errors is slightly diminished, as compared for example to SECDED. However, the tolerance to temporal MBEs is kept within an acceptable range. On the other hand, a CPPC is capable of correcting spatial multi-bit errors with a very small overhead and has some very desirable features. One of these features is to decouple the protections of dirty data and clean data. The other attractive feature of CPPC is to offer the flexibility of different levels of reliability which can be chosen based on the requirements of the system design. The level of reliability can be increased by adding register pairs and/or parity bits.

5.1 Area

A CPPC adds error correction capability to a parity protected cache at a small area cost. For example, if a cache has one parity bit per word and dedicates a small area to protection codes, we can add correction to it by adding only two registers and two barrel shifters which is a negligible area overhead. Other schemes such as SECDED codes have more area overhead.

5.2 Performance

The main purpose of caches is to improve performance. In a CPPC extra read operations (read-before-write) increase the pressure on the read port of an L1 cache, and, in some processors, may affect the scheduling of instructions. However, this operation is only required on a Store to a dirty word. Furthermore, a CPPC is very efficient as an L2 cache where the effect of this read-before-write operation on performance is negligible. By contrast, a two-dimensional parity-protected cache needs to perform read-before-write operations for *all* Stores and for *all* misses, which exacerbates the port contention problem. The downsides of SECDED ECC which is widely used in commercial processors are the long decoding latency and the area and energy overhead.

5.3 Energy consumption

A critical advantage of the CPPC scheme over other cache reliability schemes is that the spatial multi-bit error correction capability can be expanded at a very small energy cost. For example, the correction capability of a CPPC for spatial MBEs can be doubled from 4x4 squares to 8x8 squares by simply doubling the number of parity bits while its dynamic energy consumption remains almost unchanged. By contrast, the energy consumption of SECDED with bit interleaving to tolerate spatial

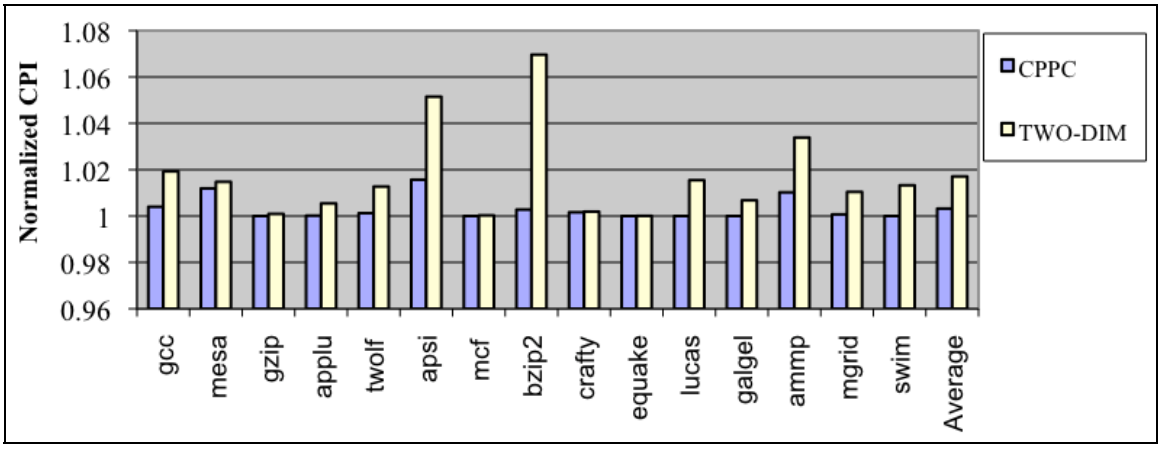


Figure 10. CPIs of Processor with CPPC and Two-Dimensional Parity L1 Caches Normalized to the CPI of Processor with One-Dimensional Parity-Protected Cache

multi-bit errors grows rapidly as the interleaving degree increases. This is a very important point as ITRS predicts that in 2016 we will only have spatial multi-bit errors and there will be no single bit error [10].

6. EVALUATIONS

In this section we compare the following caches through detailed simulations.

CPPC: As an L1 cache, a CPPC has two registers, R1 and R2, and eight interleaved parity bits per word with the byte-shifting technique. As an L2 cache, a CPPC has eight interleaved parity bits per block and two registers with the size of an L1 block and exploits the byte-shifting technique as well.

One-dimensional parity cache: As an L1 cache, this cache is protected by eight parity bits per word and does not correct dirty data. As an L2 cache, each block is protected by 8 interleaved parity bits.

SECDED cache: As an L1 cache, a word-level SECDED code is combined with 8-way data bit interleaving. As an L2 cache, a SECDED is attached to a block instead of each word.

Two-dimensional parity cache: This cache is protected by 8-way horizontal interleaved parity bits per word and per block for L1 and L2 caches (respectively), and it has one row of vertical parity to correct errors.

These configurations have been chosen in such a way that they have almost similar area and spatial multi-bit error correction capabilities, except for the two-dimensional parity protected cache. In our simulations of the two-dimensional parity cache only one vertical parity row is implemented for the entire cache so that it has similar hardware requirements as the CPPC configuration. With a single vertical parity row, the two-dimensional parity scheme loses its correction capability against multi-bit errors since it needs eight vertical parity rows to correct all 8-bit spatial errors. Given this, for a fair comparison, we do not compare the reliability of the two approaches. Rather we only compare their performance and energy consumption.

We compare the four caches under various criteria using analytical models, and SimpleScalar and CACTI simulations. We execute 100 million instructions Simpoints of the Spec2000 benchmarks [21] compiled for the Alpha ISA. Table 1 lists the parameters of the simulations.

6.1 Performance

In our simulations, we chose the same access latency of two cycles for both one-dimensional parity and SECDED caches. Hence, we have supposed that the decoding latency of SECDED is not in the critical path as data can be read without checking the protection code. To detect errors, SECDED is checked in the background and if there is an error, the read operation is repeated.

Table 1. Evaluation Parameters

Parameter	Value
Processor	
Functional Units	4 integer ALUs 1 integer multiplier/divider 4 FP ALUs 1 FP multiplier/divider
LSQ Size / RUU Size	16 Instructions / 64 Instructions
Issue Width	4 instructions / cycle
Frequency	3 GHZ
Cache and Memory Hierarchy	
L1 data cache	32KB, 2-way, 32 byte lines, 2 cycles latency
L2 cache	1MB unified, 4-way, 32 byte lines, 8 cycles latency
L1 instruction cache	16KB, 1-way, 32 byte lines, 1 cycle latency
Other Parameters	
Feature Size	32nm

Figure 10 shows the CPI of processors with CPPC and two-dimensional parity L1 caches. Port contention and access latency of the caches are simulated to determine the CPIs. As is shown in Figure 10, the performance overhead of CPPC over the one-dimensional parity protected cache is 0.3% on the average and at most 1% across all benchmarks. The performance overhead of two-dimensional parity is on average 1.7% and up to 6.9%. We compare the performance for L1 caches only because the performance effects of these protection schemes are more pronounced in an L1 cache than in an L2 cache.

6.2 Energy consumption

To compute the dynamic energy consumption of all caches, we count the number of read *hits*, write *hits*, and read-before-write operations in the various caches. The dynamic energy consumption of each operation is estimated by CACTI. In CPPC, a read-before-write is needed on every Store to a dirty word but, in the two-dimensional parity cache, it is needed on all Stores and on all misses filling clean cache lines. We do not count the energy

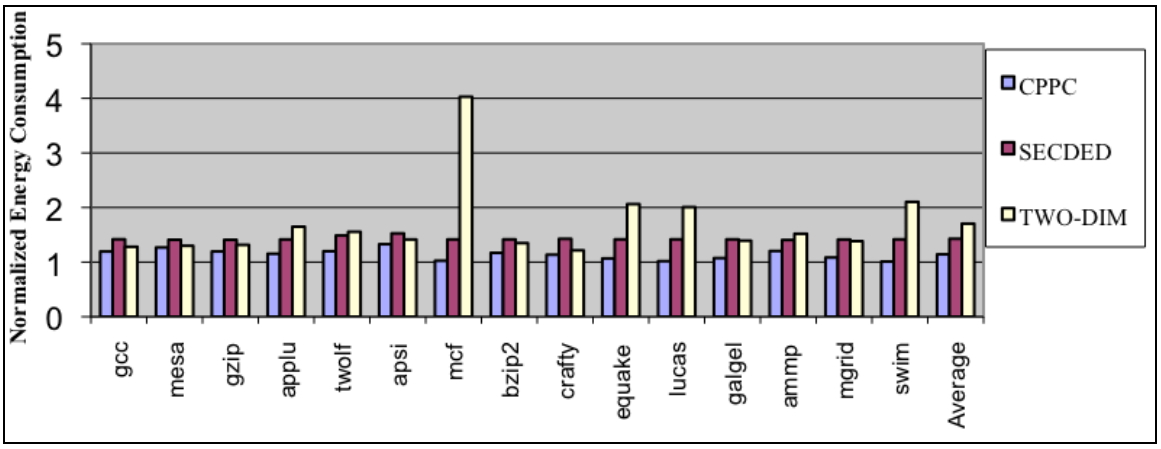


Figure 11. Dynamic Energy Consumption in Various L1 Caches Normalized to that of One-Dimensional Parity-Protected L1 Cache

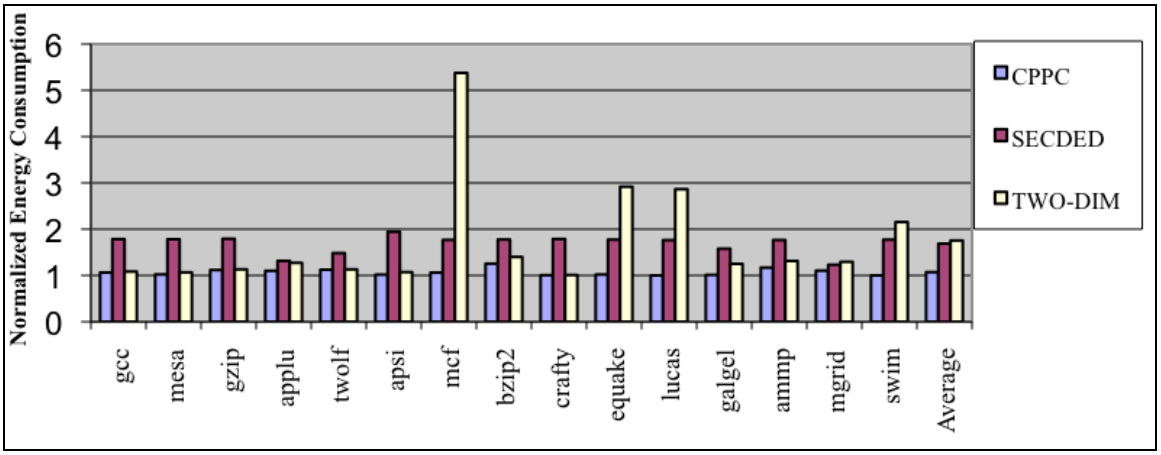


Figure 12. Dynamic Energy Consumption in Various L2 Caches Normalized to that of One-Dimensional Parity-Protected L2 Cache

spent in write-back operations. For interleaved SECDED, we multiply the energy consumption of bitlines by eight and add it to the energy consumption of the cache as it increases the number of precharged bitlines eight times [12]. Figure 11 shows the dynamic energy consumption of the various L1 cache options normalized to one-dimensional parity. As is shown in Figure 11, the energy consumption of CPPC is on average 14% higher than the one-dimensional parity cache. However, the energy consumed in other schemes is much higher. Two-dimensional parity increases energy consumption by an average of 70%, and SECDED increases energy consumption by an average of 42%.

We also compare CPPC, SECDED, two-dimensional parity and one-dimensional parity in the context of an L2 cache. Although one-dimensional parity is not common in L2 caches, we use it as the baseline to normalize energy consumption in order to show how CPPC is relatively more energy efficient in lower levels of the cache hierarchy. Figure 12, shows the energy consumption of different L2 caches normalized to the energy consumption of one-dimensional parity. The L2 CPPC consumes only 7% more energy than the one-dimensional parity cache while it provides both single-bit and multi-bit error correction capabilities for dirty data. The difference between the relative energy consumptions in L1 and L2 CPPCs is due to fewer read-before-write operations in the L2 cache. By comparison, the dynamic energy consumption of SECDED and two-dimensional parity L2 caches are 68% and 75% higher than one-dimensional parity. The energy consumption of two-dimensional parity is several times than that of CPPC for benchmark Mcf because Mcf experiences many cache misses. The L2 miss rate of Mcf is about 80% in our experiments.

6.3 Reliability

In this section, we compare the reliability of the cache options against temporal multi-bit errors. All of the compared caches except for the one-dimensional parity cache tolerate almost the same amount of spatial multi-bit errors, thus we do not consider spatial multi-bit errors in this section. Rather we focus on the impact of various protection schemes on temporal MBEs.

One of the features of CPPC is that it enlarges the domain of protection. A CPPC with eight parity bits in effect has eight protection domains whose size is 1/8 of the entire dirty data. By contrast, the protection domain in a SECDED cache is a word or a cache block. Our goal in this section is to show that since the SEU rate is very small, increasing the size of the protection domain has very little effect on the reliability against temporal multi-bit errors.

We compute the average percentage of dirty data in both L1 and L2 caches by simulating benchmarks. Table 2 shows the average percentage of dirty data across all 15 benchmarks used in Section

6. The average percentage of dirty data is obtained for each benchmark and the average over all benchmarks is given in Table 2. In addition to the percentage of dirty data, our reliability model needs the average time between two consecutive accesses to a dirty word in L1 or to a dirty block in the case of an L2 cache. We call this access time Tavg in this paper. Table 2 also gives the average of Tavg for both L1 and L2 caches across the 15 benchmarks.

The MTTF of the one-dimensional parity cache is calculated as the expected time for a fault to occur in the dirty data of the cache multiplied by 1/AVF. The Architectural Vulnerability Factor (AVF) is the probability that a fault will affect the result of the program. We use the average percentage of dirty data across all benchmarks to compute the MTTF.

Table 2. Parameters Used in Reliability Evaluations

Cache	L1	L2
Percentage of dirty data during program execution (%)	16	35
Average "Tavg" of all benchmarks (cycle)	1828	378997

Table 3. MTTF of the Caches (Temporal MBEs Faults)

Cache	MTTF of L1 caches	MTTF of L2 caches
One-dimensional parity	4490 Years	64 years
CPPC	$8.02 * 10^{21}$ Years	$8.07 * 10^{15}$ Years
SECDED	$6.2 * 10^{23}$ Years	$1.1 * 10^{19}$ Years

CPPC reliability evaluation is very challenging as the vulnerability of a bit depends not only on the interval between two accesses to it, but also on accesses to other dirty words. Hence, it requires a new approach to provide reliability estimates. Moreover, CPPC has error correction capability which must be considered in the reliability evaluation. To have a failure in CPPC, two single-bit faults should occur in a protection domain (1/8 of dirty data) and the second fault should occur before the correction of the first one. We use the new approximate analytical model introduced in [22] which was shown to have good accuracy. To be consistent, we use the same model for SECDED. In this model, the probability (P) of having two faults during Tavg, which is non-correctable, is calculated using the model [22]. The expected number of intervals (1/P) before the occurrence of two faults during Tavg, multiplied by 1/AVF yields the MTTF. For the SECDED-protected cache, the MTTF is the time before the occurrence of two faults in one dirty word in the L1 cache or in one dirty block in the L2 cache and is computed in the same way as for CPPC.

Based on this model, Table 3 shows the MTTF of different cache options for temporal MBEs, under the following assumptions. We suppose that the SEU rate is 0.001 FIT/bit in which a FIT is equal to one failure per billion hours (here failure means bit flip). Since our computations in this section concentrate only on dirty data and all Loads from dirty data may cause a failure, we assume an AVF of 70%. It is clear that the reliability of one-dimensional parity drops dramatically as the size of the cache increases. Thus, in large caches, parity without correction is not acceptable. As an L1 cache, a CPPC improves the MTTF very much as compared to one-dimensional parity and provides a very high level of resiliency to temporal multi-bit faults. As Table 3 shows CPPC is highly reliable as an L2 cache, too. Although the reliability of SECDED is better than CPPC, the reliability of CPPC is already so high that it can safely be used in critical systems.

The results of this section demonstrate that error correction can be implemented more efficiently by enlarging the protection domain because reliability does not suffer much (as in CPPC).

7. CONCLUSIONS AND FUTURE WORK

CPPC is a new reliable write-back cache which increases the granularity of error correction from words or blocks to larger granularities up to the entire cache in order to save resources such as area and energy. In CPPC, the granularity of protection can be adapted to the desirable amount of reliability. Moreover, CPPC decouples the protection of clean data from dirty data in order to assign fewer resources to the clean data in the cache.

To tolerate spatial multi-bit faults CPPC does not require physical bit interleaving which greatly increases cache energy consumption. The byte-shifting scheme proposed in this paper is added to the basic CPPC to tolerate spatial multi-bit errors. Instead of the byte-shifting technique, CPPCs can also be built with more pairs of registers to correct all spatial multi-bit errors with negligible overheads.

CPPC is quite efficient in L1 caches and its efficiency is even better in L2 caches. Based on our experimental results, CPPC adds only 7% energy overhead and practically no other overheads to an L2 parity-protected cache while it provides both single-bit and multi-bit error correction for dirty data.

We expect an L3 CPPC to be even more energy efficient. This will be explored in our future work by running new benchmarks with larger memory footprints. We believe that the number of read-before-write operations is smaller in L3 caches. Additionally, we will exploit CPPC in multiprocessors to consider the effect of write invalidate or update coherence protocols on the scheme. In invalidate protocols, since many dirty blocks may be invalidated, the number of read-before-write operations might decrease which might lead to better efficiency in multiprocessor CPPCs.

We will also evaluate single-ported caches and their impact on the read-before-write operations, in addition to some new techniques which can execute the Load part of read-before-write at a small cost in L1 caches.

Finally, the approach used for data in CPPC can be extended to cache tags. For the tags, the concept of dirty vs. clean data does not exist. Read-before-write operations are not needed. Tags are read-only until they are replaced. Therefore, the extension of the ideas developed in this paper to cache tags (including state bits) is an interesting and valuable future pursuit.

8. ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation under Grants No. CSR-0615428 and CCF-0834798. Mehrtash Manoochehri was supported by a USC Viterbi school of engineering Dean's fellowship.

We thank Jinho Suh for his thoughtful technical comments. We also thank anonymous referees for their beneficial suggestions. In particular Guri Sohi gave us valuable technical feedback, which improved the technical content of the paper significantly.

9. REFERENCES

[1] Ando, H., Seki, K., Sakashita, S., Aihara, M., Kan, R., Imada, K., Itoh, M., Nagai, M., Tosaka, Y., Takahisa, K., and Hatanaka, K. Accelerated Testing of a 90nm SPARC64 V Microprocessor for Neutron SER. In Proceedings of *IEEE Workshop on Silicon Errors in Logic - System Effects (SELSE)*, April 2007.

[2] Asadi, G., Sridharan, V., Tahoori, M. and Kaeli, D. Balancing Performance and Reliability in the Memory Hierarchy. in the Proceedings of *International Symposium on Performance Analysis of Systems and Software*, pp. 269-279, 2005.

[3] Bertozzi, D., Benini, L. and DeMicheli, G. Error Control Schemes for On-Chip Communication Links: The Energy–Reliability Tradeoff. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 24, Issue 6, pp. 818-831, 2005.

[4] Burger, D. C. and Austin, T. M. The SimpleScalar tool-set. Version 2.0. *Technical Report* 1342, Dept. of Computer Science, UW, June 1997.

[5] CACTI 5.3. DOI = http://quid.hpl.hp.com:9081/cacti/

[6] Cortex R series processors. DOI = http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0363e/Chdgfjac.html

[7] Dweik, W., Annavaram, M., and Dubois, M. Reliability Aware Exceptions. USC EE-Computer Engineering Technical Report, CENG 2011-2.

[8] Genua, P. Error Correction and Error Handling on PowerQUICC™ III Processors. DOI = http://www.freescale.com/files/32bit/doc/app_note/AN3532.pdf

[9] Huntzicker, S., Dayringer, M., Soprano, J., weerasinghe, A., Harris, D., Patil, D. Energy-delay tradeoffs in 32-bit static shifter designs. In *IEEE International Conference on Computer Design (ICCD)*, pp. 626-632, 2008.

[10] ITRS 2007 Edition. DOI = http://www.itrs.net/links/2007itrs/2007_Chapters/2007_SystemDrivers.pdf, pp. 22-23.

[11] Kessler, R. The Alpha 21264 Microprocessor. *IEEE Micro*, Vol. 19, Issue 26, pp. 24-36, 1999.

[12] Kim, J., Hardavellas, N., Mai, K., Falsafi, B. and Hoe, J. C. Multi-bit Error Tolerant Caches Using Two Dimensional Error Coding. in the Proceedings of the *40th Annual ACM/IEEE International Symposium on Microarchitecture (MICRO-40)*, pp. 197-209, 2007.

[13] Kim, S. Area-Efficient Error Protection for Caches. In the proceedings of *Design, Automation and Test in Europe*, pp. 1-6, 2006.

[14] Kongetira, P., Aingaran, K., and Olukotun, K. Niagara: A 32-way multithreaded Sparc processor. *IEEE Micro*, Vol. 25, Issue 2, pp. 21-29, 2005.

[15] Li, L., Degalahal, V., Vijaykrishnan, N., Kandemir, M. and Irwin, M. J. Soft Error and Energy Consumption Interactions: a Data Cache Perspective. in the Proceedings of *International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 132-137, 2004.

[16] Maiz, J., Hareland, S., Zhang, K. and Armstrong, P. Characterization of Multi-bit Soft Error Events in Advanced SRAMs. *IEEE International Electron Devices Meeting*, pp. 21.4.1- 21.4.4, 2003.

[17] McNairy, C. and Soltis, D. Itanium 2 processor micro architecture. *IEEE Micro*, vol. 23, Issue 2, pp. 44–55, 2003.

[18] Quach, N. High availability and reliability in the Itanium processor. *IEEE Micro*, Vol. 20, Issue 5, pp. 61-69, 2000.

[19] Sadler, N. N., and Sorin, D. J. Choosing an Error Protection Scheme for a Microprocessor's L1 Data Cache. in *International Conference on Computer Design (ICCD)*, pp. 499-505, October 2006.

[20] Seongwoo, K. and Somani, A. K. Area Efficient Architectures for Information Integrity in Cache Memories. in *International Symposium on Computer Architecture (ISCA)*, pp. 246-255, 1999

[21] Simpoint. DOI= http://cseweb.ucsd.edu/~calder/simpoint/points/standard/spec2000-single-std-100M.html.

[22] Suh, J., Manoochehri, M., Annavaram, M. and Dubois, M. Soft Error Benchmarking of L2 Caches with PARMA. in the proceedings of *ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2011.

[23] Yoon, D. H., and Erez, M. Memory Mapped ECC: Low-Cost Error Protection for Last Level Caches. In *International Symposium on Computer Architecture (ISCA)*, pp. 83-93, 2009.

[24] Zhang, W., Gurumurthi, S., Kandemir, M. and Sivasubramaniam, A. ICR: In-Cache Replication for Enhancing Data Cache Reliability. in the Proceedings of *International Conference on Dependable Systems and Networks (DSN)*, pp. 291-300, 2003.

[25] Zhang, W. Replication Cache: a Small Fully Associative Cache to Improve Data Cache Reliability. *IEEE Transactions on Computers*, Vol. 54, Issue 12, pp. 1547-1555, 2005.